D1.4

Version: 1.0
Date: 2010-12-29
Dissemination status: PU
Document reference: D1.4

Runtime Environment

Project acronym:	COMPAS
Project name:	Compliance-driven Models, Languages, and Architectures for Services
Call and Contract:	FP7-ICT-2007-1
Grant agreement no.:	215175
Project Duration:	01.02.2008 – 28.02.2011 (36 months)
Co-ordinator:	TUV Technische Universitaet Wien (AT)
Partners:	CWI Stichting Centrum voor Wiskunde en Informatica (NL)
	UCBL Université Claude Bernard Lyon 1 (FR)
	USTUTT Universitaet Stuttgart (DE)
	TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant (NL)
	UNITN Universita degli Studi di Trento (IT)
	TARC-PL Telcordia Poland (PL)
	THALES Thales Services SAS (FR)
	PWC Pricewaterhousecoopers Accountants N.V. (NL)

This project is supported by funding from the Information Society Technologies Programme under the 7th Research Framework Programme of the European Union.

Project no. 215175

COMPAS

Compliance-driven Models, Languages, and Architectures for Services

Specific Targeted Research Project

Information Society Technologies

Start date of project: 2008-02-01 Duration: 36 months

D1.4 Runtime Environment

Revision 1.0

Due date of deliverable: 2010-12-31

Actual submission date: 2010-12-29

Organisation name of lead partner for this deliverable:

USTUTT – Universitaet Stuttgart, Germany

Contributing partner(s):

TUV – Technische Universitaet Wien, Austria

UNITN – University of Trento, Italy

TARC-PL – Telcordia Poland, Poland

Project funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

History chart

Issue	Date	Changed page(s)	Cause of change	Implemented by
0.1	2010-05-20	All sections	New document, initial structure, draft of content	USTUTT
0.2	2010-06-17	Structure has been revised, Abstract has been written and first version of section on Process Engine Environment has been composed	Preparation of discussion on the contribution of the involved COMPAS partners	USTUTT
0.25	2010-07-01	Structure has been revised and advices regarding future content and partner contributions have been adapted due to feedback and comments from discussion during meeting in Pergine Valsugana	Final preparation of initial version for agreement with involved partners on future content and expected partner contributions	USTUTT
0.3	2010-07-13	Comments and improvement suggestions from USTUTT internal review of first draft have been integrated	Preparation of final version of first draft	USTUTT
0.35	2010-07-20	Subsections of sections 4.6 and 4.7 have been combines	Comments during finding of an agreement on final version of first draft	USTUTT
0.4	2010-09-26	Subsection 4.4.4 has been updated. First version of contribution by TARC-PL has been integrated (Subsections 4.6 and 4.8.1)	Update of optional prefiltering in Compliance Custom Controller. Integration of initial version of contribution from TARC-PL	TARC-PL
0.45	2010-10-06	Sections 3 and 4 have been updated.	Updates and extensions in preparation of 10th quarterly COMPAS meeting	USTUTT, TARC-PL
0.5	2010-11-08	Section 4.2	Update the generation of BPEL and deployment configuration with VbMF	TUV
0.51	2010-11-21	Section 4.1	MORSE information retrieval	TUV
0.6	2010-11-21	Section 4.2	Revise the section, replace new figures	TUV
0.7	2010-11-26	All sections	Prepared for internal review	USTUTT
0.8	2010-12-13	All sections	Revision	USTUTT, TUV, UNITN
0.81	2010-12-17	All sections	Prepared for approval	USTUTT
1.0	2010-12-24		Approval	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	USTUTT	2010-12-17
2	Approved	TUV	2010-12-24
3	Released	TUV	2010-12-29

Contents

List of figures

List of listings

Abstract

This deliverable introduces the integrated COMPAS runtime environment which enables compliance checking and monitoring at runtime. The runtime environment considers storage of compliance-related process data and the enactment of process execution including traceability aspects. The compliance-enabled service environment publishes execution events via an enterprise service bus. Several components are subscribed to the published events to enable online and offline compliance monitoring. The results of the monitoring are displayed in a compliance governance dashboard. This deliverable contains integration code for prototypes developed in the project (e.g., code for connecting publishers and subscribers for the enterprise service bus). In addition, we present a new prototype for online process instance monitoring which provides basic monitoring functionality missing in the standard distribution of the process engine.

1. Introduction

1.1. Purpose and scope

This deliverable provides a prototype for checking the compliance rules for the runtime compliance environment. The runtime environment consists of different components, which enable checking compliance rules at runtime and analyzing violations.

1.2. Document overview

Section 2 describes the positioning of this deliverable in the overall COMPAS architecture. Section 3 briefly describes the motivation and concept of traceability and its impact on the COMPAS runtime components. Section 4 presents the integration code for each component and provides references to the source code and binaries of the tools. Section 5 contains references to internal and external documents.

1.3. Definitions and glossary

The most important terminology concerning the COMPAS project is listed on the public COMPAS Web-Site [D7.1] available at http://www.compas-ict.eu, section “terminology”. This helps to make the overall COMPAS approach more comprehensive for the general public.

In the following the definitions of terms valid only in the scope of this deliverable (and therefore not listed on the public COMPAS Web-Site) are specified. To offer a self-contained deliverable, general terms of the COMPAS terminology are copied here.

Compliance requirement: A Compliance requirement is a constraint or assertion that results from the interpretation of compliance sources.

Compliance fragment: A Compliance fragment (i.e., a process fragment for compliance) is a connected process structure that can be used as a reusable building block to ensure a consistent specification of compliance regarding a business process. Compliance fragments can be used to implement a compliance rule in terms of activities and control structures.

Compliance target: A Compliance target is a generic specification, such as a business process, or a compliance fragment, which is a target of compliance requirements.

Traceability: Traceability in the field of compliance denotes the property to be able to trace a compliance requirement throughout the lifecycle of artefacts. An artefact in COMPAS is an activity, a variable, a correlation set of a process, a process model itself, a process instance, execution events, etc.

1.4. Abbreviations and acronyms

BPEL	Business Process Execution Language
CEP	Complex Event Processing
CLRT	Compliance Language Request Tools
DSL	Domain-Specific Language
MDD	Model-Driven Development
MDSD	Model-Driven Software Development
MORSE	Model-Aware Repository & Service Environment
oAW	openArchitectureWare
ODE	Orchestration Director Engine
SQL	Structured Query Language
UUID	Universally Unique Identifier
VbMF	View-based Modelling Framework
WSDL	Web Services Description Language
XML	Extensible Markup Language

2. Integration into the COMPAS architecture

This chapter describes which of the components of the COMPAS overall architecture are affected and involved in the compliance monitoring and checking at runtime. This includes the mechanism how the traceability is achieved. Note that we focus in this deliverable mainly on the components contained within the runtime compliance environment as well as the compliance governance architecture. The basis for traceability is provided by the MDSD Software Framework when generating the BPEL processes as well as the configurations for components. Such components are the Compliance Custom Controller (Java Properties File) and the CEP Engine (CEP rules).

Figure 1 depicts the overall COMPAS architecture and highlights the components affected in this deliverable. The model repository deals with storage and traceability information (see Section 4.1). The centrepiece of the execution is the process engine which executes business processes by means of service orchestration. During execution of a process, execution events are emitted by the process engine and the services which are invoked. These events are published to an Enterprise Service Bus (ESB), which forms the integration platform between runtime and monitoring. The online monitoring components subscribe to the ESB in order to

receive these events. The complex event processing engine also generates higher level events, which are themselves published to the ESB. The event log stores the events which are published on the ESB in order to enable offline monitoring.

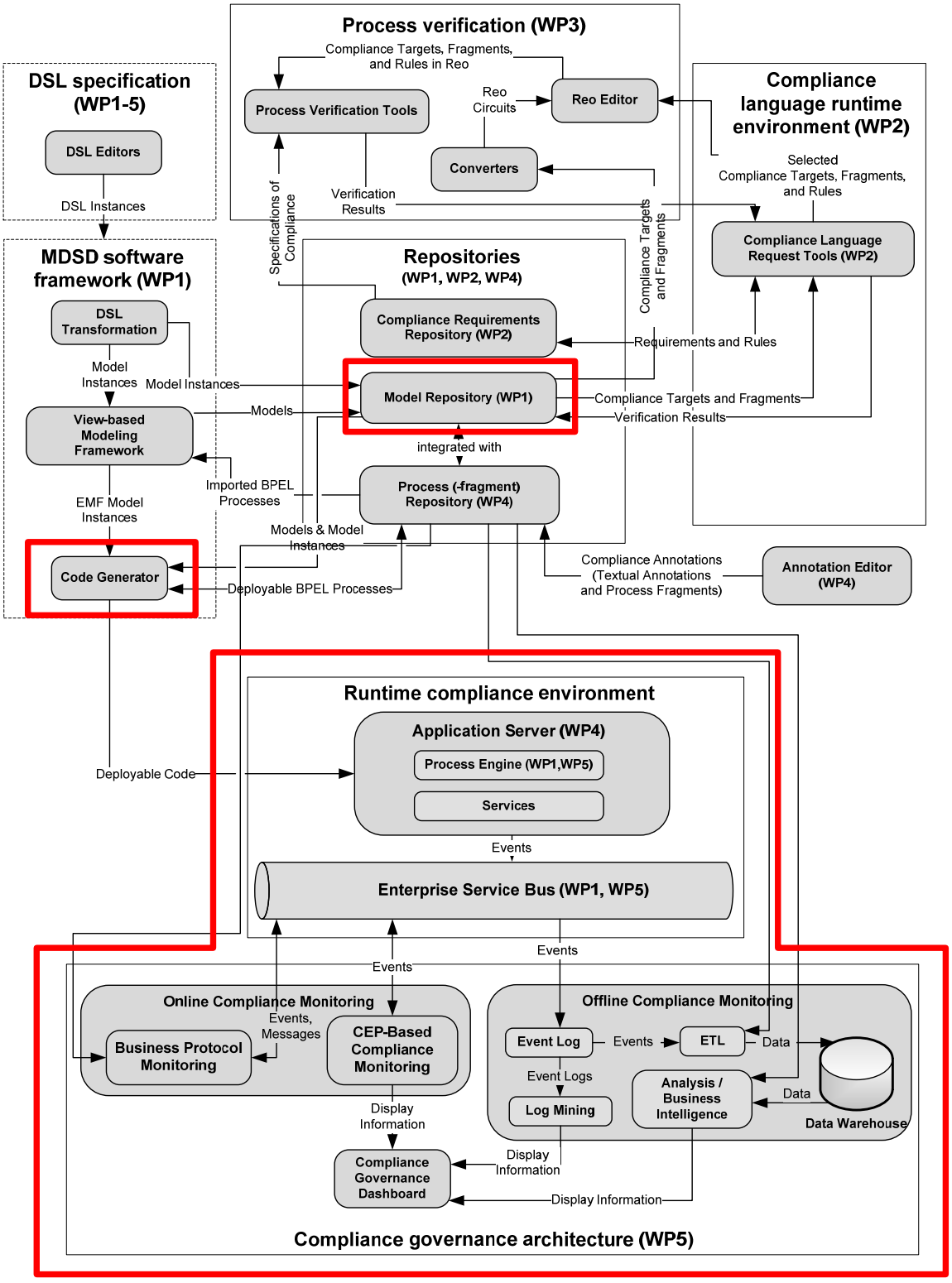

Figure 1 Overall COMPAS architecture

3. Concept and application of traceability

Figure 2 presents an overview on the concept of traceability. It shows from left to right a high-level process model, a low-level process model, an engine-internal model, execution events and a dashboard. A traceability information is identified by an ID and shown as a red ID in the figure. In the high-level process model, a single activity is annotated with an ID. From a compliance point of view, this identifier is related to a compliance requirement. This ID is used in the whole lifecycle of the process. The low-level process is an executable process model used for deployment at a process engine. New annotations may be made at this stage (blue IDs). Possibly, the engine also injects identifiers to the deployed process model stored using an internal format. All these IDs are contained in the events published by the engine. These events are used at a compliance dashboard to enable drill-down of compliance violations to the root cause. In summary, it is fundamental to support traceability in a compliance-enabled runtime environment.

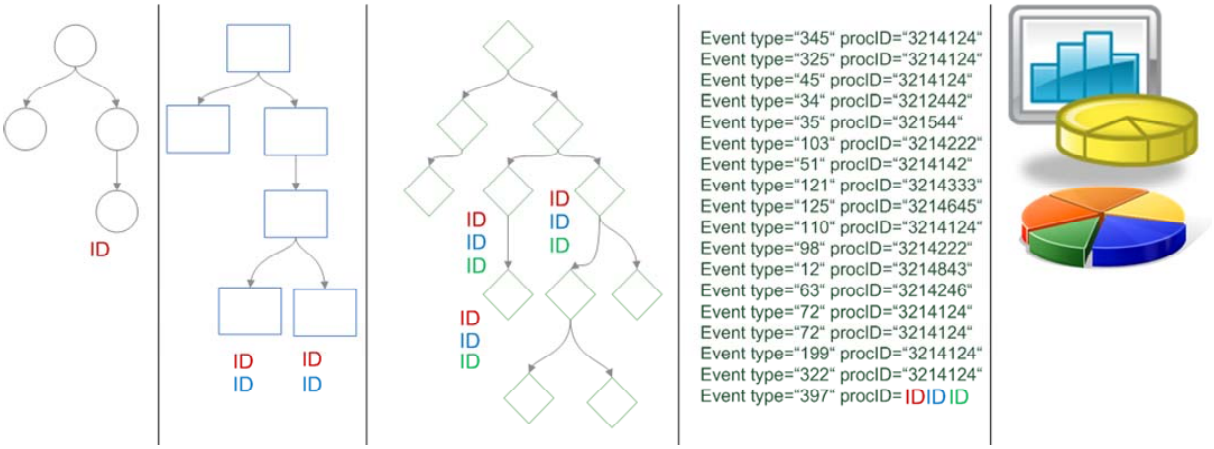

Figure 2 Overview on traceability

With respect to the COMPAS components traceability information is injected into the processes using universally unique identifiers (UUIDs). This happens at the level of BPEL processes. Listing 1 shows an excerpt of the traceability information for the WatchMe use case [D5.4, D6.3]. The code presented in Listing 1 assigns a unique identifier to an activity.

```
<morse:traceability
   xmlns:morse="http://xml.vitalab.tuwien.ac.at/ns/
      morse/traceability.xsd"
   build="56810150-5bd8-4e8e-9ec5-0b88a205946b"
   xmlns="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">
  <row query="/process/sequence[1]/pick[3]/onMessage[1]
              /sequence[1]/if[2]/elseif[1]/sequence[1]/flow[1]
              /sequence[1]/invoke[1]">
     <uuid>ab12349a-iu1z-007a-1z5z-1xy56df11654</uuid>
  </row>
</morse:traceability>
```

Listing 1 Example for traceability information

4. Integrated COMPAS tools for checking compliance at runtime

Figure 3 presents the technical architecture of the components described in this deliverable. The figure shows the exchange of artefacts (black arrows), the deployment of processes and configurations (red arrows) as well as the flow of events (blue arrows). The upper part of the figure contains the components of the MDSD software framework and the repositories. The middle part makes up the runtime compliance environment. It consists of an extended Apache ODE process engine, a compliance custom controller for emission of events augmented with traceability information, and an enterprise service bus (Apache ActiveMQ). The lower part consists of the components of the compliance governance architecture. The architecture is split up into online and offline monitoring. The following subsections describe the corresponding integration code for each of these groups of components.

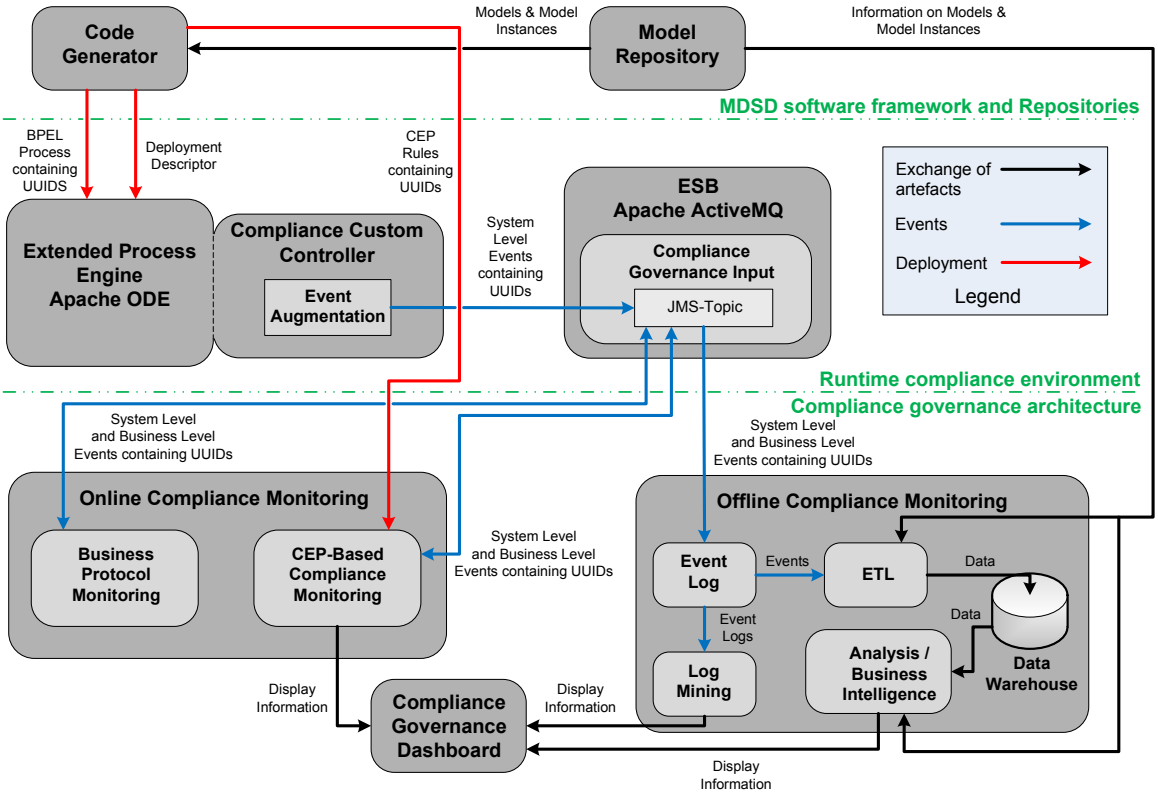

Figure 3 Integrated COMPAS architecture for compliance checking at runtime

4.1. Compliance information retrieval

All models and model instances that are used for generating the model-driven SOA (e.g., compliance targets) or relate to concepts from the COMPAS conceptual model (e.g., compliance requirements) are retrievable from the MORSE model repository by any (runtime, design-time) component interested. This is, besides interfaces for the storage of instances from the View-based Modelling Framework and the COMPAS conceptual model, the MORSE repository particularly offers service-based interfaces for the retrieval of these to runtime monitoring components (see also Table 1 from [D4.4b]).

Besides these XML and RESTful Web service interfaces the MORSE also automatically generates ready to use Java clients (i.e. proxies). These are packaged and distributed via a Maven repository and can, e.g., be included as a Maven dependency, in the client software.

The code example presented in Listing 2 uses the MORSE library for the COMPAS conceptual model and retrieves a compliance requirement for a given UUID.

```
CRequirementServiceImpl impl = new CRequirementServiceImpl(uuid);
CRequirement req = null;
try {
      req = impl.retrieve();
} catch (RemoteException e) {
      System.err.println("Some error while communication with MORSE!");
} catch (CRequirementNotFoundException e) {
      System.err.println("Compliance Requirement not found!");
}
```

Listing 2 Java-example on how to retrieve a compliance requirement

The code example displayed in Listing 3 uses the MORSE library for the COMPAS conceptual model. For a given compliance target, identifiable by the UUID `uuidTarget`, it demonstrates how to retrieve all compliance requirements that are related to these compliance targets via controls. In this example the `query` operation is called with following parameters: `head`, `jpaJoin`, `jpaWhere`, `jpaOrder`, `firstResult`, and `maxResults`. If the first parameter is `false` then version specific copies are queried otherwise so called version independent copies. The MORSE repository realizes a transparent model versioning so that identifiers can be reused for retrieving the most recent version of a model. In addition to such a version independent copy of the model, the MORSE repository also stores version specific copies of the model as the runtime usually needs to relate to a specific version of a model. Thus, for runtime component it usually makes sense to query an exact version; however if the runtime needs to consider the most recent, e.g., compliance requirement, this parameter can be set to `true`. The second parameter specifies the JOIN clause of a JPA Query (JPQL) (cf. [JPA09]). Similarly the two following optional parameters specify the WHERE and ORDER clause of such a query. Finally, for pagination `firstResult` and `maxResults` can be set to zero or specific numbers (cf. JPA 2.0 specification [JPA09]), e.g., for only returning the fifth and sixth result the parameters would be specified as `5-1` and `2`.

```
Collection<ControlImpl> controls = null;
try {
      controls = new ControlServiceImpl().query(
true, "o.targets t" , "t.uuid = '" + uuidTarget + "'", null, 0, 0);
} catch (RemoteException e) {
      System.err.println("Some error while communication with MORSE!");
} catch (ControlNotFoundException e) {
      System.err.println("There are no Controls for the process!");
}
EList<CRequirement> reqs = new BasicEList<CRequirement>();
for (ControlImpl control : controls) {
      reqs.addAll((Collection<? CRequirement>) control.getRequirements());
}
// reflect on all Requirements
for (CRequirement req : reqs) {
      // reflect on the compliance Requirement: req
}
```

Listing 3 Java-example on how to retrieve all compliance requirements that are related to a certain compliance target

4.2. Process generation and deployment

Figure 4 shows the Eclipse-based user interface of the View-based Modeling Framework (VbMF) for generating process implementation in BPEL/WSDL as well as the deployment configuration and runtime directives for traceability. VbMF also offers the deployment of BPEL processes into Apache ODE engine [Apache09a]. Firstly, the option "…with deployment files" in the code generation dialog (see Figure 4) must be checked such that the deployment configuration for the Apache ODE engine will be generated accordingly. Secondly, the users can configure corresponding VbMF preferences (see Figure 5) to let VbMF know where the Apache ODE engine is hosted. Finally, users just go to menu "VbMF" and choose "Deploy".

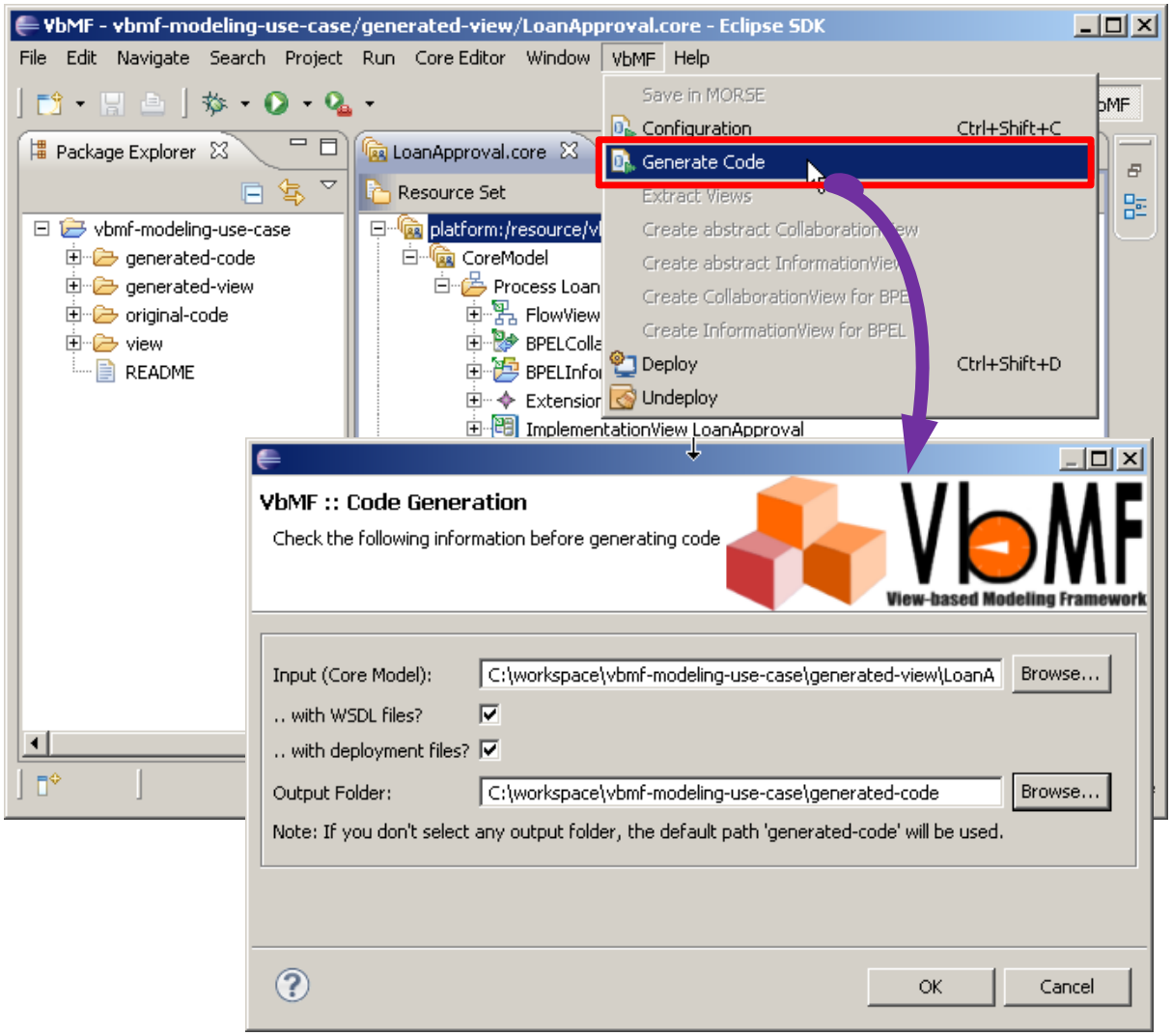

Figure 4 VbMF supports for generating BPEL/WSDL and configuration

In addition, users can also configure the generation of traceability elements in the aforementioned preference page.

VbMF is one of the main contributions of [D1.2] and has been implemented in [D1.3]. Therefore, we opt to not elaborate further in this document but rather recommend the readers to consult [D1.2, D1.3] for more details.

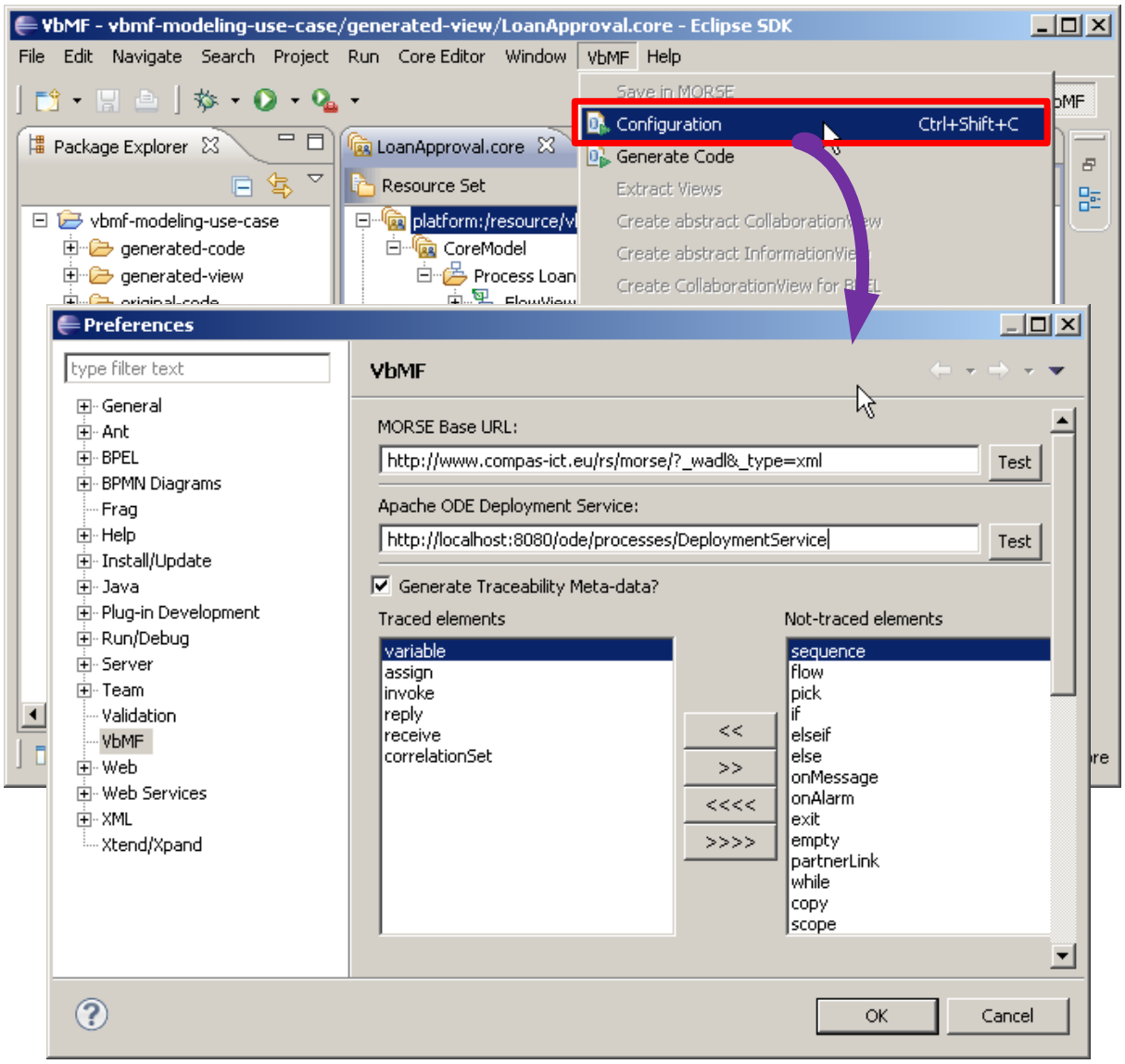

Figure 5 VbMF Preference Page for deployment and traceability configuration

4.3. Process execution and event publishing

The extended process engine consists of a modified version of Apache ODE and a compliance custom controller. The Apache ODE has been extended to support the "pluggable framework for enabling the execution of extended BPEL behaviour" [KKL07]. This framework provides the generation of generic BPEL events, which are independent of the BPEL engine used. These events provide information of status changes of activities and variables. A so-called "custom controller" can connect to the engine in order to receive and process such events. We have implemented such a custom controller and call it "compliance custom controller". Details are described in [D4.4b]. The compliance custom controller generates events with traceability information (Section 3) and publishes them to the Enterprise Service Bus (Section 4.4), where they are consumed and used to perform online compliance monitoring (Section 4.6) and offline compliance monitoring (Section 4.8). The traceability information enables the monitoring components to retrieve additional information from the Model Repository for drill down in case a compliance violation has been detected, see Figure 3.

Each BPEL process has to be enriched by traceability information (Section 3). BPEL allows the process definition to be extended with elements not defined in the BPEL specification. The extension has to be declared in an <extensions> element. The extension itself may be put at the beginning of each element. We added the traceability information at the beginning of the <process> element. Listing 4 shows an example for a BPEL process definition with traceability information. This process definition is sent along with a "process deployed" event to the compliance custom controller, which stores the traceability information to enable the addition of UUIDs to the events generated by the engine.

```
<bpel:process name="WatchMeProcess" targetNamespace="http://www.compas-
ict.eu/watchme">
  <morse:traceability
xmlns:morse="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
xmlns="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
build="c3fc8f18-50db-4589-b296-26b5132650a3">
    <row query="/process"
queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      <uuid>4c546447-db90-41e0-a3c5-f82229e55346</uuid>
      <uuid>ba3e61a2-60e2-4c25-a080-44ada7e2d9cf</uuid>
      <uuid>8f964426-12f8-4845-8d1d-dc4d44707c02</uuid>
      <uuid>5c9ff665-fa02-4ead-8256-c58e2cf9c79a</uuid>
      <uuid>a8526580-c373-4676-944e-67a5e04f1a63</uuid>
      <uuid>7b54edf8-3acd-4823-a0a2-c084a71a7c44</uuid>
      <uuid>4da0069b-b4de-4f9a-bb2a-9d3967be4e63</uuid>
    </row>
    <row
query="/process/sequence[1]/pick[1]/onMessage[1]/sequence[1]/invoke[1]">
      <uuid>50b9f4d6-624c-35ba-a617-1736dee0dc17</uuid>
    </row>
    <row
query="/process/sequence[1]/pick[1]/onMessage[1]/sequence[1]/if[1]/sequence
[1]/invoke[1]">
      <uuid>491a73fa-d3cf-3599-9921-3da57e479885</uuid>
    </row>
    <row
query="/process/sequence[1]/pick[1]/onMessage[1]/sequence[1]/if[1]/sequence
[1]/reply[1]">
      <uuid>3ba56ccf-105f-3486-8d53-0729750eb7e4</uuid>
    </row>
    ...
  </morse:traceability>
  ...
  <bpel:extensions>
    <bpel:extension mustUnderstand="no"
namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"/>
  </bpel:extensions>
```

Listing 4 Excerpt of BPEL process file augmented with traceability information

Listing 5 shows an example of an execution event which is emitted by the compliance custom controller. The event denotes the start of the execution of the activity invoking the service responsible for checking the user data. This event is sent to the enterprise service bus, where the monitoring components use it for compliance monitoring. The topic used is named "eu.compas_ict.events.lowlevel".

```
<Event xmlns="http://xml.infosys.tuwien.ac.at/ns/compas/event">
  <timestamp>2010/11/22 23:02:33</timestamp>
  <source>Process Engine</source>
  <type>ActivityExecutingEvent</type>
  <name>ac0af35f-45a5-4907-ba02-a67d1abc573e</name>
  <processBuildUUID>268a3509-bbca-4640-a00a-9ab2ca2592c1
  </processBuildUUID>
  <processInstanceId>3251</processInstanceId>
  <property name="messageID">27</property>
  <property name="activityName">Check user data</property>
  <property xmlns:ns3="http://www.compas-ict.eu/watchme"
              name="processName">ns3:WatchMeProcess
  </property>
  <property name="processUUIDList">
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">5a3b5
7ac-ac45-4381-8f79-0853ab90e58a
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">f2be4
55d-aceb-4c8c-9eca-0c71d24f4ed0
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">445f2
8ff-0be5-4019-ae30-e03cccfbea13
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">594bb
ecc-5df2-4afb-9f13-4b3e54381f9f
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">13878
820-c310-493c-8d21-4409dfcdbc1c
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">5cd1d
8ea-d0ef-44c7-b6dc-16adbaec9418
    </ns2:uuid>
    <ns2:uuid
xmlns:ns2="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd">c2c53
4c0-2af1-4afd-a14f-fb8d006d965a
    </ns2:uuid>
  </property>
  <property name="processVersion">5</property>
  <property name="activityInstanceID">3302</property>
  <property
name="activityXPath">/process/sequence[1]/pick[1]/onMessage[1]/sequence[1]/
invoke[1]
  </property>
  <property name="scopeInstanceID">3301</property>
  <property name="scopeXPath">/process</property>
  <property name="uuid">50b9f4d6-624c-35ba-a617-1736dee0dc17</property>
  <property xmlns:ns="http://www.compas-ict.eu/watchme"
name="portType">ns:UserDataCheck
  </property>
  <property name="operation">checkUserData</property>
  <property name="endpointReference">
    <service-ref xmlns="http://docs.oasis-open.org/wsbpel/2.0/serviceref">
     <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
          <Metadata>
```

```
                    <ServiceName
xmlns="http://www.w3.org/2006/05/addressing/wsdl"
                          xmlns:servicens="http://www.compas-ict.eu/watchme"
EndpointName="UserDataCheck">servicens:UserDataCheckService
                    </ServiceName>
                </Metadata>
                <Address>
http://localhost:8080/UserDataCheckWS/services/UserDataCheck
                </Address>
            </EndpointReference>
        </service-ref>
    </property>
</Event>
```

Listing 5 Example of an execution event emitted

The compliance custom controller is part of the compliance runtime environment. It has to be integrated with the components of the compliance governance architecture, namely Event Log, Business Protocol Monitoring and CEP-Based Compliance Monitoring (cf. Figure 3). This integration is described here. We use the publish/subscribe messaging paradigm and use the Java Messaging Service to integrate the components. Apache ActiveMQ [Apache09b] is used as JMS provider. Details are described in Section 4.4.

The compliance custom controller publishes events to a JMS topic. The events are augmented with traceability information and contain information required for compliance checking. The JMS topic is created by the compliance custom controller (Listing 6).

The UUID identifying the build process as well as the UUIDs uniquely identifying the views that have been used during generation at design time are contained in all events emitted by the compliance custom controller.

```
// variables for setup of topic
// used to send messages to the governance architecture

private ConnectionFactory factory;
private Connection outConnection;
private Session outSession;
private Topic outTopic;
private MessageProducer outPublisher;
try {
   factory = new ActiveMQConnectionFactory("tcp://localhost:61616");
   outConnection = factory.createConnection();
   outSession = outConnection.createSession(false,
      Session.AUTO_ACKNOWLEDGE);
   outTopic = outSession.createTopic("eu.compas_ict.events.lowlevel");
   outPublisher = outSession.createProducer(outTopic);
   outPublisher.setDeliveryMode(DeliveryMode.PERSISTENT);
   System.out.println("Connection to Apache ActiveMQ established
      successfully!");
}
catch (Exception e){
   System.out.println("Establishing connection to Apache ActiveMQ
      failed!");
   System.out.println(e);
}
```

Listing 6 Java source code for creation of the JMS topic

```
public void sendMessageToGovernanceArch(Serializable aObject){
   try {
      ObjectMessage aObjectMessage = outSession.createObjectMessage();
      aObjectMessage.setObject(aObject);
      outPublisher.send(aObjectMessage);
      System.out.println("Message has been sent to topic of governance
         architecture including corresponding uuids from traceability
         information.");
   }
   catch (JMSException e){
      System.out.println("Message could not be send to Topic.");
      System.out.println(e);
   }
   catch (Exception e){
      System.out.println("Exception while sending message to Topic.");
      System.out.println(e);
   }
}
```

Listing 7 Java source code for publishing an event message to the JMS topic

The compliance custom controller subscribes to the JMS topic created by the extended Apache ODE. The subscription is similar to the one used to be used subscribe to the topic created by the compliance custom controller. The description of subscribing to a topic is provided in Section 4.4.

4.4. Subscription to the Enterprise Service Bus

In COMPAS we use the open source Apache ActiveMQ [Apache09b] as messaging infrastructure. Creation of JMS topics and the sending of messages has been described in Section 4.3. In this section, we show the subscription to a JMS topic and how messages can be processed.

Listing 8 presents an excerpt of the XML schema describing the format of each published event. The full XML schema is provided with the source code.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:evt="http://xml.infosys.tuwien.ac.at/ns/compas/event"
targetNamespace="http://xml.infosys.tuwien.ac.at/ns/compas/event">
      <xs:element name="Event">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="evt:timestamp"/>
                        <xs:element ref="evt:source"/>
                        <xs:element ref="evt:type"/>
                        <xs:element ref="evt:name"/>
                        <xs:element ref="evt:processBuildUUID"/>
                        <xs:element ref="evt:processInstanceId"/>
                        <xs:element ref="evt:property" minOccurs="0"
                                    maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:simpleType name="EventType">
      <xs:restriction base="xs:string">
      <xs:enumeration value="ActivityCompleteEvent"/>
      <xs:enumeration value="ActivityExecutedEvent"/>
      <xs:enumeration value="ActivityExecutingEvent"/>
```

```
...
      <xs:enumeration value="VariableModificationEvent"/>
      <xs:enumeration value="VariableReadEvent"/>
      <xs:enumeration value="ComplianceViolationEvent"/>
      </xs:restriction>
      </xs:simpleType>
      <xs:element name="name" type="xs:string"/>

      <xs:element name="timestamp" type="xs:string"/>

      <xs:element name="source" type="xs:string"/>
      <xs:element name="processBuildUUID" type="xs:string"/>
      <xs:element name="processInstanceId" type="xs:string"/>
      <xs:element name="type" type="evt:EventType"/>
      <xs:element name="property">
            <xs:complexType mixed="true">
                  <xs:sequence>
                        <xs:any namespace="##any" processContents="lax"
                              minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:anyAttribute namespace="##other"
                                    processContents="lax"/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Listing 8 XML Schema of Events (Excerpt)

Listing 9 shows the subscription to a topic. First of all, the URL of the ActiveMQ broker has to be provided. We put ActiveMQ's failover transport on top of the tcp transport in order to be able to deal with inactivity timeouts: ActiveMQ closes the connection after a certain period of time. As compliance monitoring has to be always available, the connection should never be closed. This is enabled by the failover transport. The class used to handle each incoming message is set by the method "setMessageListener".

```
// variables for subscribing to the topic
private TopicConnectionFactory mTopicConnectionFactory;
private TopicConnection mCommunicationInConnection;
private Topic mCommunicationInTopic;
private TopicSubscriber mCommunicationInSubscriber;
private TopicSession mCommunicationInSession;
try {
   // JMS- Topic the Compliance Custom Controller events are published to -
      Topic for incoming messages for all interested components of
      compliance governance architecture
   mTopicConnectionFactory =
      new
ActiveMQConnectionFactory(("failover:(tcp://localhost:61616)?wireFormat.max
InactivityDuration=0,timeout=3000,maxReconnectAttempts=65535");
   mCommunicationInConnection =
      mTopicConnectionFactory.createTopicConnection();
   mCommunicationInSession =
      mCommunicationInConnection.
      createTopicSession(false,TopicSession.AUTO_ACKNOWLEDGE);
   mCommunicationInTopic =
      mCommunicationInSession.createTopic("eu.compas_ict.events.lowlevel");
   mCommunicationInSubscriber =
      mCommunicationInSession.createSubscriber(mCommunicationInTopic);
   // register MessageDispatcher for Topic
```

```
    MsgDispatcher messageDispatcher = new MsgDispatcher();
    mCommunicationInSubscriber.setMessageListener(messageDispatcher);
    mCommunicationInConnection.start();
    System.out.println("Connection to Apache ActiveMQ established
       successfully!");
}
catch (Exception e) {
    System.out.println("Establishing connection to Apache ActiveMQ
       failed!");
    System.out.println(e);
}
```

Listing 9 Java source code for subscribing to a JMS topic

Listing 10 presents the method being called at each receipt of a message. In COMPAS, XML-based event are used [D5.4]. As a consequence, each received event has to be converted to a Java-Object and then processed or directly processed using an XML processor. In Listing 10, the XML processor StAX [Codeh10] is chosen to process the received message. In COMPAS, several components are subscribed to the execution events emitted by the process engine, for instance the Complex Event Processing (CEP) and the Event Log,

```
public void onMessage(Message message) {
      TextMessage msg = (TextMessage) message;
      String xmlText = null;
      try {
            xmlText = msg.getText();
      } catch (JMSException e) {
          System.out.println("Error occurred during serialization of
            received message!");
          System.out.println(e);
      }
      // parse XML using your favorite parsing framework
      // here we use StAX
      try {
            XMLInputFactory inputFactory = XMLInputFactory.newInstance();
            InputStream in = new
              ByteArrayInputStream(xmlText.getBytes("UTF-8"));
            XMLEventReader eventReaderUnfiltered =
              inputFactory.createXMLEventReader(in);
            XMLEventReader eventReader = eventReaderUnfiltered;
            while (eventReader.hasNext()) {
                  XMLEvent event = eventReader.nextEvent();
                  // handle event here
            }
      } catch (Exception e) {
            e.printStackTrace();
      }
}
```

Listing 10 Java source code for handling messages received from a JMS topic after subscription

4.5. Online process instance monitoring

The Business Process Illustrator (BPI) is a web-based tool for monitoring the execution of business processes. It displays the current state of a process instance. The process graph is refreshed regularly. The tool has been developed in a Diploma Thesis, supervised by

USTUTT [Lat10]. The source code, binaries, and installation manual are available for download at SourceForge [BPI10].

In [SLS10a] we discussed the foundations of views on processes. In [SLS10b] we built on this knowledge, and described how views on processes can be used to support compliance management. BPI builds on this research and implements concepts which are related to compliance management in business processes. With respect to business process monitoring, these are the highlighting and hiding of compliance fragments. Another important feature of this tool beyond regular process monitoring is the omission of structures which are of little importance for understanding, e.g., assign or validate activities. This feature is important when dealing with complex processes containing a large number of activities. BPI is integrated with the BPEL process engine Apache ODE, which we extended to support traceability, as discussed in [D4.4b].

For selection of a process to be monitored, the user may choose from the selection menu shown in Figure 6. This menu allows selecting, searching and filtering for particular process models which are deployed on the process engine (upper part), and selecting corresponding process instances (lower part).

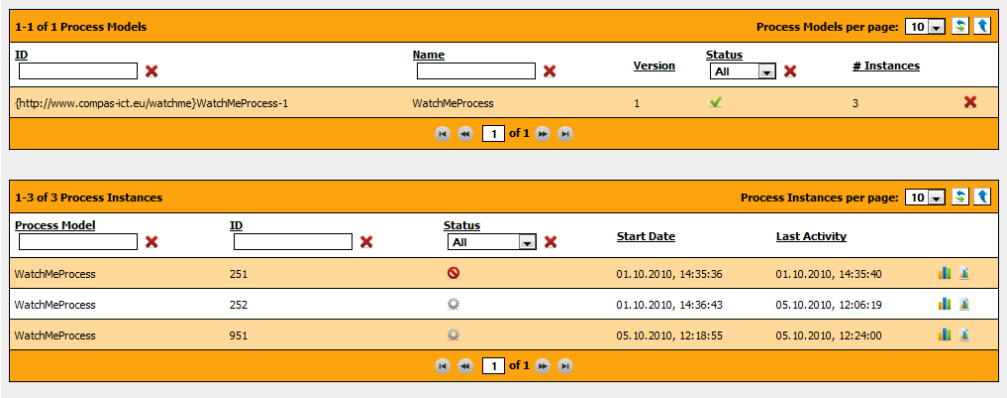

Figure 6 Selection menu for deployed processes and corresponding instances

After a selection of a process instance has been made, the process diagram is generated using Scalable Vector Graphics (SVG), which is subsequently displayed in the client's browser. BPI supports several viewing modes for process diagrams: the full mode shows for each activity the start and end time, the activity type, name, status and a corresponding icon, as shown in Figure 7. The process diagram is updated in pre-defined refreshment cycles. Thus, the process diagram always reflects the current state of the process instance.

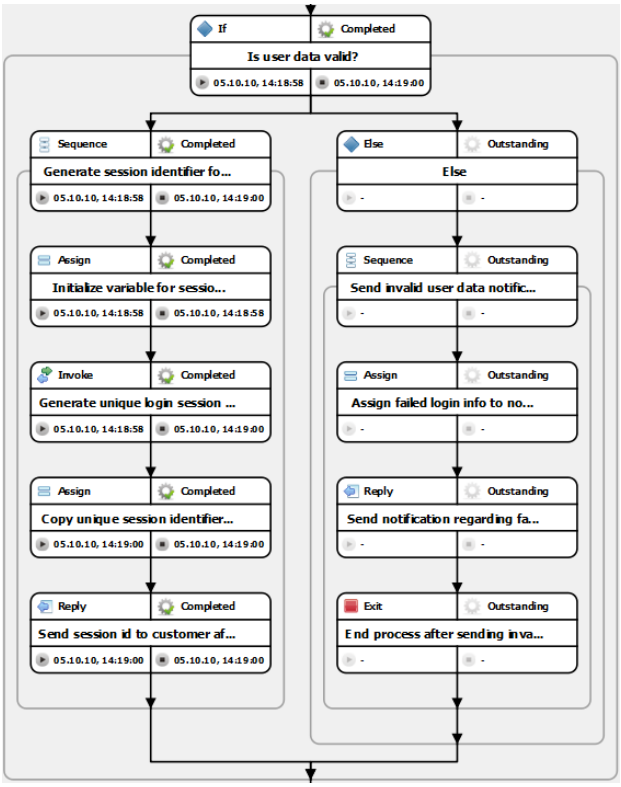

Figure 7 Excerpt of the full mode of the business process diagram for WatchMe

The compact mode reduces the amount of information displayed, however for complex processes the overall complexity is still very high, see for instance the business process diagram for the WatchMe use case depicted in Figure 8.

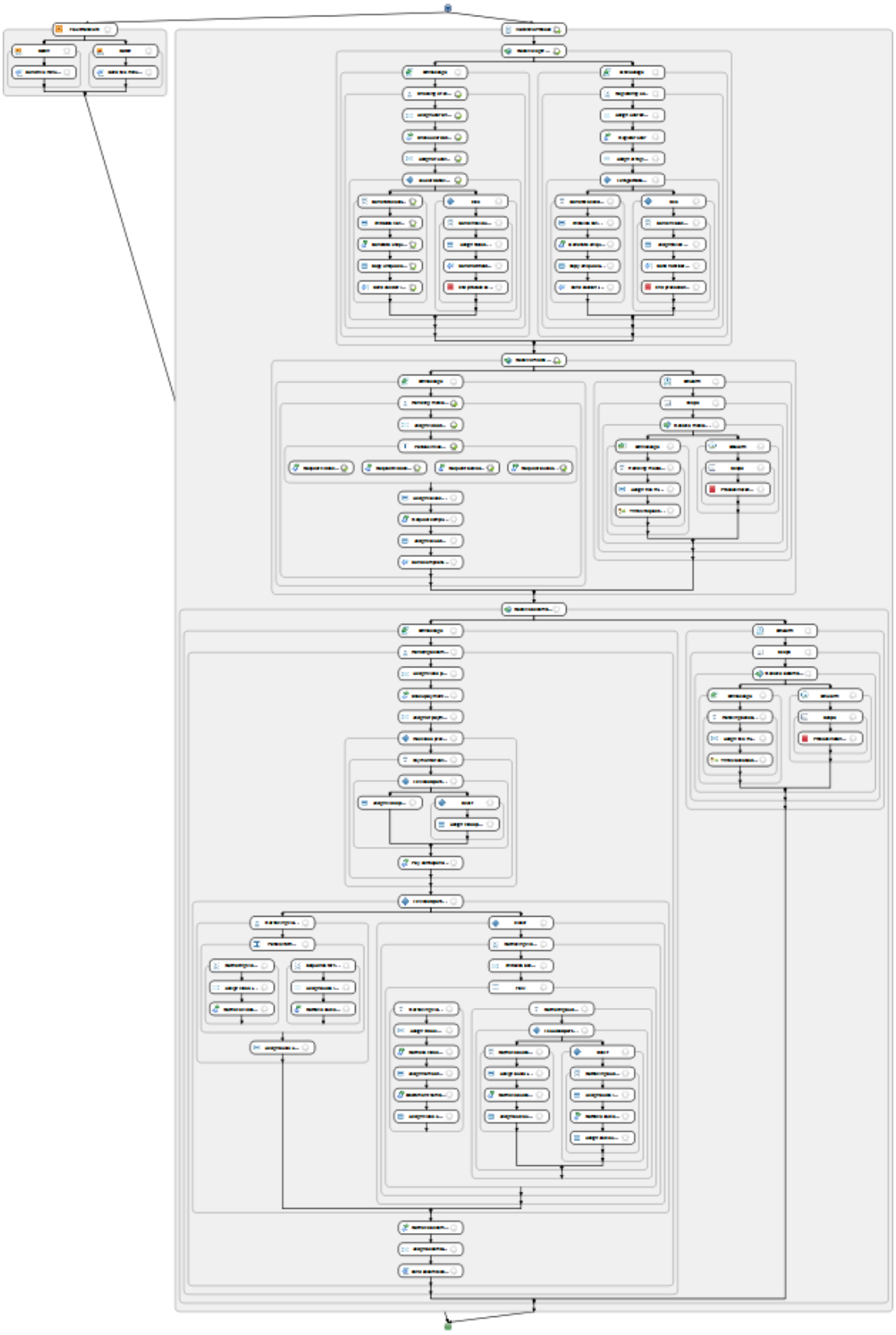

Figure 8 Compact mode of the business process diagram for WatchMe. Diagram is still too complex and requires abstraction.

In order to reduce the complexity of a process the tool supports the omission of activities which are of little importance for particular tasks. For instance, *assign* and *validate* activities can be omitted, or fault handling logic can be hidden, without impairing the overall process structure. Another feature is the possibility to highlight particular parts in a process. As we discussed in [SLS10b], such feature can be used to clearly highlight process structures which are related to compliance, i.e., *compliance fragments*. Figure 9 shows an excerpt from the process diagram of the WatchMe use case, where activities of little importance are omitted, and activities of a compliance fragment are highlighted in order to support an auditor. The compliance fragment implements the requirement "time-based plan" originating from domain licensing.

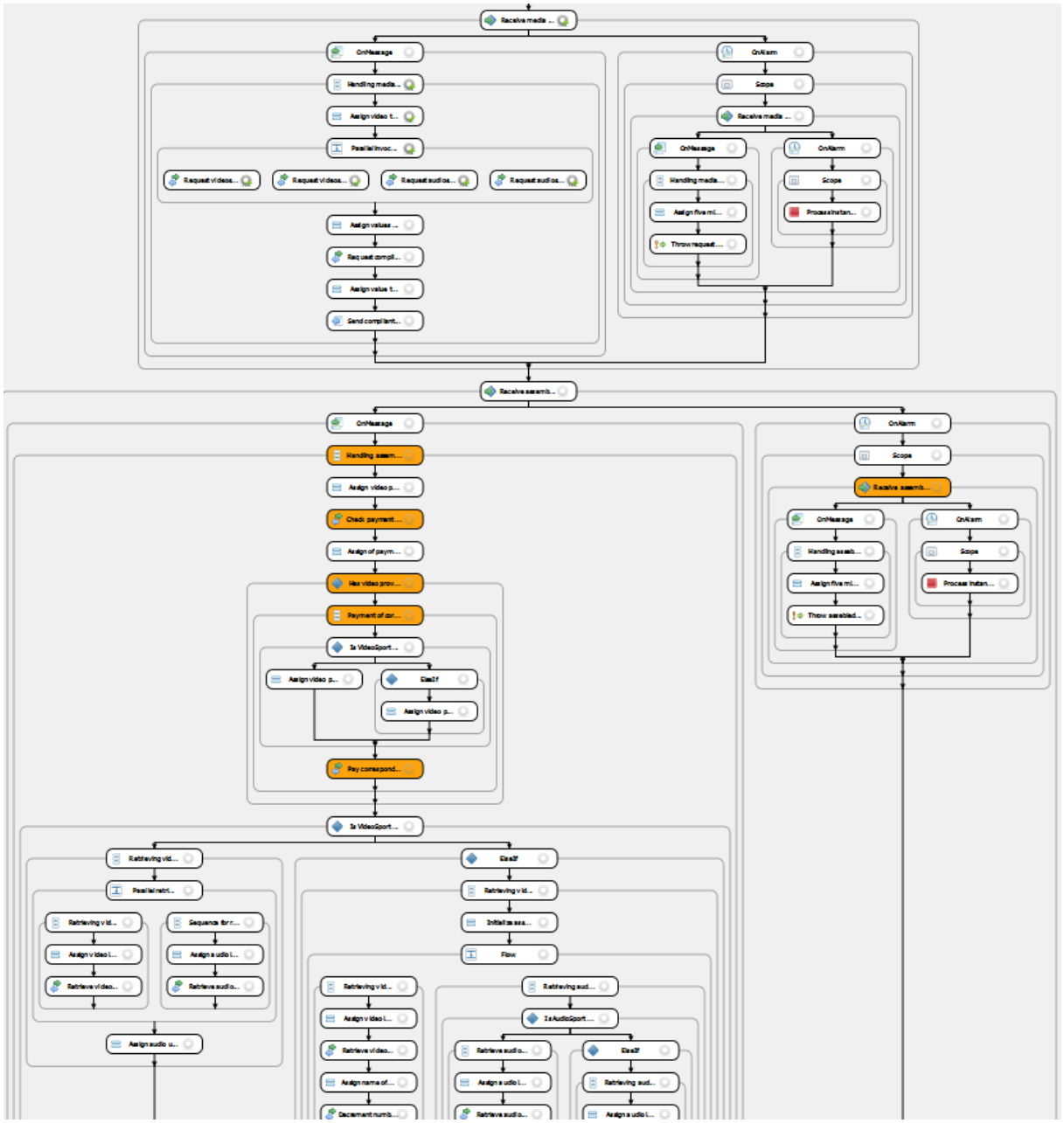

Figure 9 Excerpt from the business process diagram, where activities of little importance are omitted and activities related to time-based plan are highlighted

The tool is tightly integrated with the Apache Ode engine. It provides an adapter concept allowing it be used in other contexts.

4.6. Online compliance monitoring

Complex Event Processing (CEP) is used to enable detection of compliance violations at runtime. The CEP engine evaluates predefined compliance rules against patterns of events flowing through it. It enables matching of various event pattern templates (including time dependencies), event aggregations, filtering and various kinds of logical and arithmetic operations on event streams. The technique is based on active databases and is often called a reverse database – as instead of queries evaluated against whole relations, the flowing events are evaluated against the rules and only some events, needed for the rule to be matched, are kept in a temporary storage.

```
public class CompositionEventStmt
{
    private EPStatement statement;
    public CompositionEventStmt(EPAdministrator admin)
    {
    String stmt = "select * from " + "pattern [ every ( " +
        "V1 = Event(name='WatchMeGetVideoStreamEvent' AND " +
                  "properties.Property[2].value='1') AND " +
       "(A2=Event(name='WatchMeGetAudioStreamEvent' AND " +
                  "properties.Property[2].value='2'))" + ") " +
       "where timer:within(5000 msec)] where
      (A2.properties.Property[1].value"+"=V1.properties.Property[1].value)
  AND (A2.properties.Property[4].value = V1.properties.Property[3].value)";

      statement = admin.createEPL(stmt);
    }

    public void addListener(UpdateListener listener)
    {
        statement.addListener(listener);
    }
}
```

Listing 11 Example of Complex Event Processing rule

Listing 11 shows the source code of the class that specifies Complex Event Processing rule for monitoring the composition permission compliance requirement. The statements are specified using a special Event Processing Language (EPL) which resembles SQL. Unlike SQL, the language allows for a specification of various event patterns. Listing 11 presents an example of a pattern created with the use of simple logical AND relation.

If any of the violations patterns are detected, then a high-level notification event is generated which is directly sent to the COMPAS ESB, where it can wait for the Compliance Governance Dashboards (or other components) to be processed.

4.7. Online monitoring dashboard

This section describes the online monitoring dashboard, which is connected to the extended business process engine by consuming the events from the topic available at the ESB (cf. Section 4.4). The overall design and the prototype are delivered in the deliverable [D6.2].

A set of events or sometimes an individual event that match a single rule can cause a violation to occur. Such violations are shown on the CEP online monitoring dashboard grouped by their type and provider in real time.

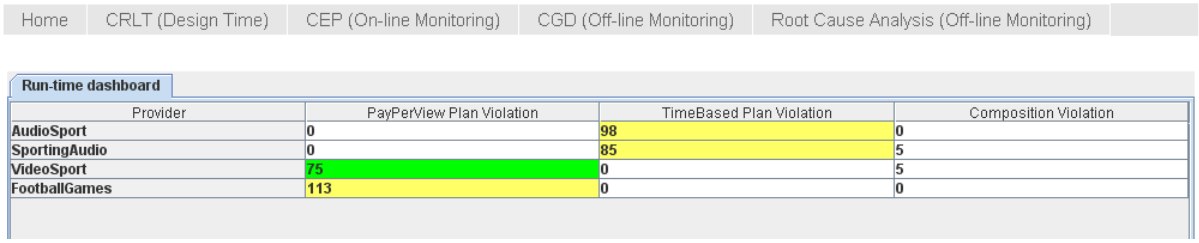

Figure 10 Realization of Online Compliance Governance Dashboard – overall view

Details of each single violation can be obtained by double-clicking one of the fields where numeric value is greater than 0 which opens up the new tab. The detailed fields vary depending on the type of violations.

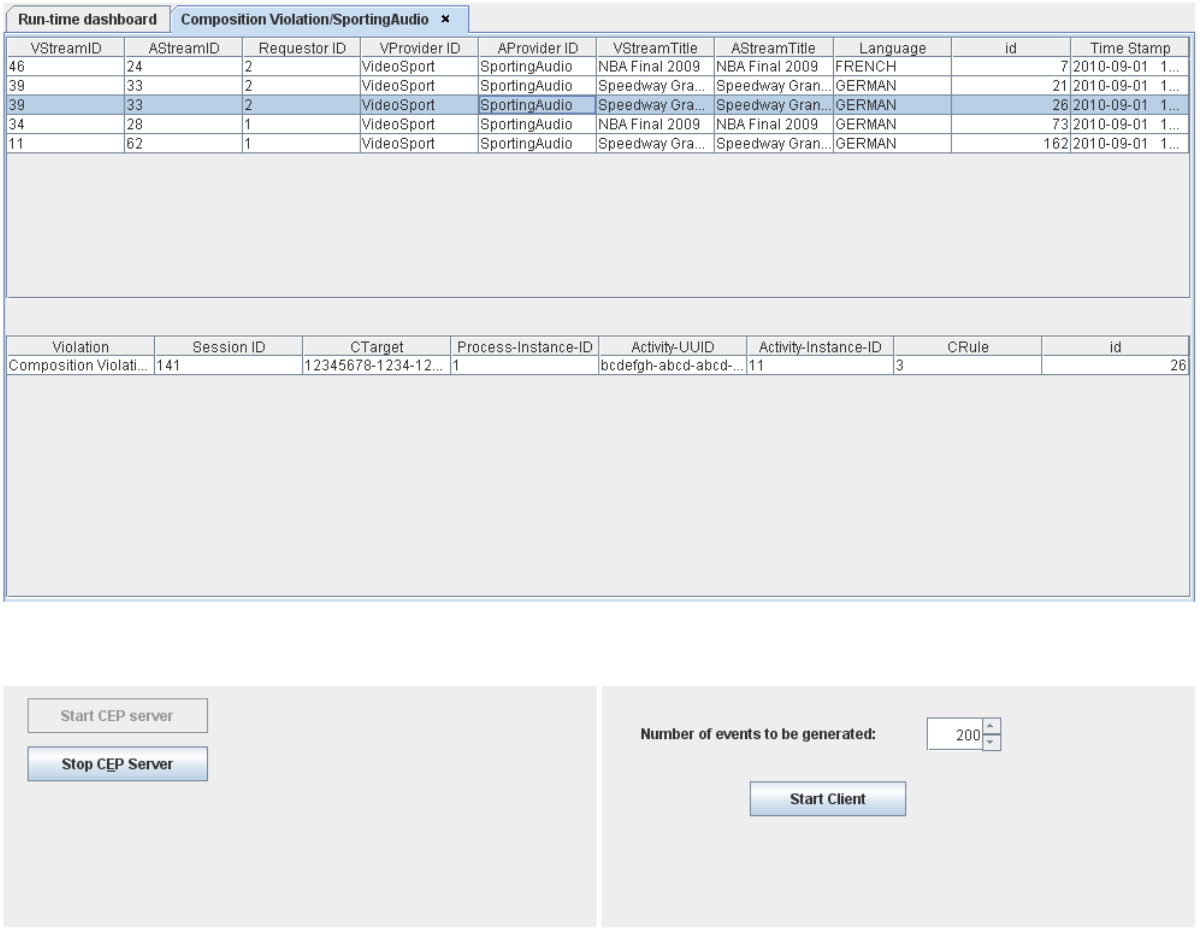

Figure 11 Realization of Online Compliance Governance Dashboard – detailed view of composition violation regarding "SportAudio" provider

The detailed view gives the information on each every single violation occurrence including timestamp, providers, process/session ID, etc. User is able to navigate between different views/sets of violations by switching tabs.

The dashboard consists of three java applets, from which two of them are emulating other parts of COMPAS architecture before the actual integration. At the top there is a main applet which listens for violation events from ActiveMQ using JMS. The dashboard is a subscriber of the topic eu.compas_ict.events.highlevel, which provides real time updates whenever a

violation happens. When the message arrives it is added to the overall view window 'Run-time dashboard' and all other tabs which correspond to this event's type.

In the Figure 10 the dashboard indicates that there were 5 composition violations corresponding to the Sporting Audio provider. A double click opens new tab with details on these particular violations. In case a new violation occurs, both of these tabs will be updated accordingly.

The first applet in the bottom of the dashboard is responsible for starting/stopping the CEP engine server. The second applet allows generating random events for testing purposes. The random event generation client is a component which connects to the ESB and sends mock events generated according to the COMPAS event model and WatchMe requirements. The corresponding applet allows for generation of arbitrary number of such events.

4.8. Offline compliance monitoring

The offline compliance monitoring, in the context of the COMPAS project, can be done according to three main approaches: Compliance Governance Dashboard (CGD), Root Cause Analysis, and Protocol Monitoring.

The CGD serves as a starting point to detect non-compliance behaviours (violations) of running process. Using it different users (e.g., CIO, external and internal auditors) have access to the most important compliance status information in order to take decision and avoid, or at least reduce, compliance risks. The design and development of the CGD are completely described in the deliverable [D5.5]. In addition, the prototype of the CGD is available on the web at http://compas.disi.unitn.it:8080/CGDs/main.jsp. The prototype supports data from the both COMPAS case study scenarios, WatchMe (Telecom CGD) and Loan Approval (Banking CGD). More information is available at the COMPAS CGD website at http://compas.disi.unitn.it/CGD/home.html.

Whenever non-compliant behaviour is detected, root-causes can be explored using the Root Cause Analysis tool. Hence, the final users can discover why some specific and frequent violations occur during the execution of a business process. Details about the design and implementation of the tool are present in the deliverables [D5.4] and [D5.5]. Moreover, the final prototype version of the Root Cause Analysis tool is available at http://compas.disi.unitn.it:8080/CGDs/main.jsp. At the moment, the tool is just working with syntactic data related to the WatchMe scenario.

In order to provide the data for the application aforementioned we developed a routine, named compliance_governance_event_receiver, to capture low-level and high-level events from the ESB and store them into the Event Log (more details in [D5.3]). The Java code used is composed of 3 main classes: Main, Communication, and Dispatcher. The first class coordinates the execution by invoking and instantiating the main objects and methods. The second subscribes to the ESB topic assigned for the compass scenarios and consumes the events. The third checks the content of the events and insert them into the main Event Log table.

Listing 12 shows the piece of code used to connect to the Event Log database, retrieve and process those events according to the destination table, and the SQL routine applied to insert them to the same table.

```
if (object instanceof ProcessDeployedMessage){
      ProcessDeployedMessage aProcessDeployedMessage =
      (ProcessDeployedMessage) object;
```

```
      Connection conn = null;
      Calendar cal = Calendar.getInstance();
      SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddhhmmss");

      try{
      OracleDataSource ds = new OracleDataSource();
      ds.setDriverType("thin");
      ds.setServerName("compas.disi.unitn.it");
      ds.setPortNumber(1521);
      ds.setServiceName("compas");
      ds.setUser("COMPAS");
      ds.setPassword("******");
      conn = ds.getConnection();
      conn.setAutoCommit(false);
      Statement stmt = conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

      String source = aProcessDeployedMessage.getSource();
      QName processName = aProcessDeployedMessage.getProcessName();
      Long timeStamp = aProcessDeployedMessage.getTimeStamp();
      String formated_timeStamp = new
        java.text.SimpleDateFormat("yyyyMMddHHmmss")
        .format(new java.util.Date (timeStamp));
      Long eventID = aProcessDeployedMessage.getMessageID();

      TextMessage body = (TextMessage)
        Communication.aXStreamMessageTransformer.consumerTransform
        (Communication.mCommunicationInSession,
         Communication.mCommunicationInSubscriber, msg);

      stmt.executeQuery("INSERT INTO log_table
       (log_id,source,process_name,timestamp_date,load_date,eventBody)
       VALUES"+
      "("+ eventID.toString() +",'"+ source +"','"+ processName.toString()
+"',(to_date('" + formated_timeStamp+ "', 'YYYY/MM/DD
HH24:MI:SS')),(to_date('" + sdf.format(cal.getTime())+ "', 'YYYY/MM/DD
HH24:MI:SS')), EMPTY_CLOB())");
      conn.close();
      } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Database access error: " + e);
      }
}
```

Listing 12 Example of compliance_governance_event_receiver code used to retrieve events from the ESB and store them into the Event Log database

5. Summary

In this deliverable we have shown the integration of the COMPAS runtime environment which enables compliance checking and monitoring at runtime. The main point of integration is the ESB. We provided integration code to create a communication channel (JMS topic), to publish execution events using this channel, and we showed how to subscribe to it. As we discussed in this deliverable, several components are subscribed to the events to enable online and offline compliance monitoring. The results of the monitoring are displayed in a compliance governance dashboard. Furthermore, we showed how the repositories and the process generation tool are integrated with this architecture.

6. Reference documents

6.1. Internal documents

[D1.2] "Core meta-models, transformation templates, and languages", 2009-12-31.

[D1.3] "MDSD software framework for business compliance – Final version", 2010-12-31.

[D4.4b] "Supporting infrastructure – process engine, process artefact repository, process generation tool", Version 1.0 of 2010-12-23.

[D5.4] "Reasoning mechanisms to support the identification and the analysis of problems associated with user requests", Version 2.1 of 2009-12-22.

[D5.5] "Final Prototype of Compliance Governance Dashboards", Version 1.0 of 2010-12-23.

[D6.2] "Application implementation and case study prototypes", Version 1.1 of 2010-07-31.

[D7.1] "Public Web-Site", http://www.compas-ict.eu

6.2. External documents

[Apache09a] Apache Software Foundation: Apache ODE (Orchestration Director Engine), http://ode.apache.org.

[Apache09b] Apache Software Foundation: Apache ActiveMQ, http://activemq.apache.org.

[BPI10] Business Process Illustrator: http://sourceforge.net/projects/bpi/

[Codeh10] Codehaus. The Streaming API for XML (StAX), http://stax.codehaus.org/

[JPA09] Java Specification Request 317: Java™ Persistence 2.0, Java Community Process, Dec. 2009, http://jcp.org/en/jsr/summary?id=317

[KKL07] R. Khalaf, D. Karastoyanova, F. Leymann: Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. In: Proceedings of the 3rd International Workshop on Engineering Service-Oriented Application (WESOA'2007), Springer, 2007.

[Lat10] G. Latuske: Sichten auf Geschäftsprozesse als Werkzeug zur Darstellung laufender Prozessinstanzen. Diploma thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3036/DIP-3036.pdf

[SLS10a] D. Schumm, F. Leymann, A. Streule: Process Viewing Patterns. In: Proceedings of the 14th IEEE International EDOC Conference (EDOC 2010).

[SLS10b] D. Schumm, F. Leymann, A. Streule. Process Views to Support Compliance Management in Business Processes. In: Proceedings of the 11th International Conference on Electronic Commerce and Web Technologies (EC-Web), Springer, 2010.

[All links were last followed on December 17, 2010.]